Capability Models for Manycore Memory Systems:
A Case-Study with Xeon Phi KNL and the COSMO Weather Code
Microarchitectures are becoming more and more complex
How to optimize codes for these complex architectures?

- **Performance engineering:** “encompasses the set of roles, skills, activities, practices, tools, and deliverables applied at every phase of the systems development life cycle which ensures that a solution will be designed, implemented, and operationally supported to meet the non-functional requirements for performance (such as throughput, latency, or memory usage).”

- **Manually profile codes and tune them to the given architecture**
  - Requires highly-skilled performance engineers
  - Need familiarity with
    - NUMA (topology, bandwidths etc.)
    - Caches (associativity, sizes etc.)
    - Microarchitecture (number of outstanding loads etc.)
An engineering example – Tacoma Narrows Bridge
Scientific Performance Engineering

1) Observe

2) Model

3) Understand

4) Build
Modeling by example: KNL Architecture (mesh)

S. Ramos and T. Hoefler: Capability Models for Manycore Memory Systems: A Case-Study with Xeon Phi KNL, IPDPS’17
KNL Architecture (memory: Flat & Cache)

S. Ramos and T. Hoefler: Capability Models for Manycore Memory Systems: A Case-Study with Xeon Phi KNL, IPDPS’17
KNL Architecture (all to all mode)

S. Ramos and T. Hoefler: Capability Models for Manycore Memory Systems: A Case-Study with Xeon Phi KNL, IPDPS'17
S. Ramos and T. Hoefler: Capability Models for Manycore Memory Systems: A Case-Study with Xeon Phi KNL, IPDPS’17
KNL Architecture (SNC-4 or SNC-2)

S. Ramos and T. Hoefler: Capability Models for Manycore Memory Systems: A Case-Study with Xeon Phi KNL, IPDPS’17
How much does this all matter?

What is the real cost of accessing cache?

What is the cost of accessing memory?
Step 1: Understand core-to-core transfers – MESIF cache coherence

Write back overhead

Location: only 5-15% difference

Contention effects?

That is curious!

All values are medians within 10% of the 95% nonparametric CI, cf. TH, RB: “Scientific Benchmarking of Parallel Computing Systems”, SC16
Step 2: Understand core-to-memory transfers – DRAM and MCDRAM

- MCDRAM 20% slower!
- MCDRAM 4-6x faster!
- Need to read **and** write for full bandwidth
- Cache mode >20% slower
- Bandwidth suffers a bit

All values are medians within 10% of the 95% nonparametric CI, cf. TH, RB: “Scientific Benchmarking of Parallel Computing Systems”, SC16
Performance engineers optimize your code!

Are you kidding me?
A principled approach to designing cache-to-cache broadcast algorithms

Multi-ary tree example

- Depth $d = 2$
- $k_1 = 2$
- $k_2 = 3$

Tree depth

$T_{tree} = \sum_{i=1}^{d} T_C(k_i) = \sum_{i=1}^{d} (c \cdot k_i + b)$

Level size

$= \sum_{i=1}^{d} (R_R + R_L + c \cdot (k_i - 1))$

Tree cost

$T_{sbcast} = \min_{d, k_i} (T_{fw} + \sum_{i=1}^{d} (c \cdot k_i + b) + \sum_{i=1}^{d} T_{nb}(k_i + 1))$

Reached threads

$N \leq 1 + \sum_{i=1}^{d} \prod_{j=1}^{i} k_j, \ \forall i < j, k_i \leq k_j$
Model-driven performance engineering for broadcast

\[ T_{sbcast} = \min_{d,k_i} \left( T_{fw} + \sum_{i=1}^{d} (c \cdot k_i + b) + \sum_{i=1}^{d} T_{nb}(k_i + 1) \right) \]

\[ N \leq 1 + \sum_{i=1}^{d} \prod_{j=1}^{i} k_j, \quad \forall i < j, k_i \leq k_j \]

Binary or binomial trees?

Oh! But sure I can do that.

S. Ramos and T. Hoefler: Capability Models for Manycore Memory Systems: A Case-Study with Xeon Phi KNL, IPDPS’17
Model-driven performance engineering for broadcast

13x faster, lower variance

Oh! But sure I can do that.

Binary or binomial trees?
Easy to generalize to similar algorithms

Barrier (7x faster than OpenMP)
Reduce (5x faster than OpenMP)
What about real applications?
A factor $2x$ in resolution roughly corresponds to a factor $10x$ compute

$\Delta x = 35 \text{ m} \ (1x)$
A factor $2\times$ in resolution roughly corresponds to a factor $10\times$ compute.

$\Delta x = 70 \text{ m (10x)}$
A factor $2x$ in resolution roughly corresponds to a factor $10x$ compute.

$\Delta x = 140 \text{ m (100x)}$
A factor $2x$ in resolution roughly corresponds to a factor $10x$ compute.

$\Delta x = 280 \text{ m} \ (1,000x)$

Image credit: Oliver Fuhrer, MeteoSwiss
A factor $2x$ in resolution roughly corresponds to a factor $10x$ compute

$\Delta x = 550 \text{ m (10,000x)}$
A factor $2\times$ in resolution roughly corresponds to a factor $10\times$ compute

**Operational model of MeteoSwiss today!**

$\Delta x = 1100 \text{ m} \ (100,000\times)$
A factor $2x$ in resolution roughly corresponds to a factor $10x$ compute.

Operational model of MeteoSwiss before 2016!

$\Delta x = 2200 \text{ m (1,000,000x)}$

We’re a factor of 100,000 away!
Basic Atmospheric Equations

\[ \begin{align*}
\frac{d\mathbf{v}}{dt} &= -\nabla p + \rho g - 2\Omega \times (\rho \mathbf{v}) - \nabla \cdot (\mathbf{T}) \\
\frac{dp}{dt} &= -(c_{pd}/c_{vd})\rho \nabla \cdot \mathbf{v} + (c_{pd}/c_{vd} - 1)Q_h \\
\frac{dT}{dt} &= \frac{dp}{dt} + Q_h \\
\rho_c \frac{dv^c}{dt} &= -\nabla \cdot \mathbf{F}^u - (\mathbf{I}^i + \mathbf{I}^f) \\
\rho \frac{dv}{dt} &= -\nabla \cdot (\mathbf{P}^l + \mathbf{F}^l) + \mathbf{I}^l \\
\rho \frac{d\mathbf{q}^I}{dt} &= \mathbf{q}^I \\
\rho &= \rho \{ R_d (1 + (R_e/R_d - 1)\mathbf{q}^g - \mathbf{q}^g - \mathbf{q}^f)T_j^{-1} \}
\end{align*} \]

Discretized on a compute grid

ECMWF-Model
- 16 km Grid
- 2 x per day
- 10 days prediction

COSMO-7
- 6.6 km Grid
- 3 x per day
- 72 h prediction

COSMO-1
- 1.1 km Grid
- 7 x pro day 33 h prediction
- 1 x pro day 45 h prediction

Slide adopted from T. Schulthess
The COSMO Code – 300k SLOC Fortran

Two main algorithmic motifs in dynamical core

Finite Difference Stencils

Tridiagonal Solvers

```fortran
    do j = 1, niter
        do i = 1, nwork
            c(i) = a(i) + b(i) * ( a(i+1) - 2.0d0*a(i) + a(i-1) )
        end do
    end do
```

k

j

i

k

j

i
Stencil computations (oh no, another stencil talk)

Motivation:
- Important algorithmic motif (e.g., finite difference method)

Definition:
- Element-wise computation on a regular grid using a fixed neighborhood
- Typically working on multiple input fields and writing a single output field

\[ \text{lap}(i,j) = -4.0 \times \text{in}(i,j) + \text{in}(i-1,j) + \text{in}(i+1,j) + \text{in}(i,j-1) + \text{in}(i,j+1) \]

Due to the typically low arithmetic intensity, stencil computations are often memory bandwidth limited!
How to tune such stencils (most other stencil talks)

- LOTS of related work!
  - Compiler-based (e.g., Polyhedral such as PLUTO [1])
  - Auto-tuning (e.g., PATUS [2])
  - Manual model-based tuning (e.g., Datta et al. [3])
  - ... essentially every micro-benchmark or tutorial, e.g.:

- Common features
  - Vectorization tricks (data layout)
  - Advanced communication (e.g., MPI neighbor colls)
  - Tiling in time, space (diamond etc.)
  - Pipelining

- Much of that work DOES NOT compose well with complex stencil programs in weather/climate

---

[1]: Uday Bondhugula, A. Hartono, J. Ramanujan, P. Sadayappan. A Practical Automatic Polyhedral Parallelizer and Locality Optimizer , PLDI'08
[3]: Kaushik Datta, et al., Optimization and Performance Modeling of Stencil Computations on Modern Microprocessors, SIAM review
What is a “complex stencil program”? (this stencil talk)

E.g., the COSMO weather code
- is a regional climate model used by 7 national weather services
- contains hundreds of different complex stencils

Modeling stencils formally:
- Represent stencils as DAGs
  - Model stencil as nodes, data dependencies as edges

simplified horizontal diffusion example

\[
\begin{align*}
a \oplus b &= \{a' + b' | a' \in a, b' \in b\}
\end{align*}
\]
Horizontal Diffusion Stencil Program tuned to Xeon Phi KNL

4th order horizontal diffusion: $-\alpha \nabla^2 (\nabla^2 u)$

2 inputs
1 output
4 stencils

3.7x improvement

Work performed at the Intel Parallel Computing Center at ETH Zurich
Vertical Advection Stencil Program tuned to Xeon Phi KNL

Vertical stencil-like operations, derived from Iterations of tridiagonal solvers (Thomas algorithm)

\[
\begin{bmatrix}
 b_1 & c_1 & 0 \\
 a_2 & b_2 & c_2 \\
 a_3 & b_3 & \ddots & c_{n-1} \\
 0 & \ddots & \ddots & \ddots & \ddots \\
 0 & 0 & \ddots & \ddots & \ddots & \ddots \\
\end{bmatrix}
\begin{bmatrix}
 x_1 \\
 x_2 \\
 \vdots \\
 x_n \\
\end{bmatrix}
=
\begin{bmatrix}
 d_1 \\
 d_2 \\
 \vdots \\
 d_n \\
\end{bmatrix}
\]

4.2x improvement

Work performed at the Intel Parallel Computing Center at ETH Zurich
Scientific performance engineering for complex memory systems

Step 2: Understand core-to-memory transfers – DRAM and MCDRAM

<table>
<thead>
<tr>
<th>Mode</th>
<th>Latency [ns] (Benchmark)</th>
<th>Software NUMA</th>
<th>Software UMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM</td>
<td>130-140</td>
<td>SNC4</td>
<td>SNC2</td>
</tr>
<tr>
<td>MCDRAM</td>
<td>160-175</td>
<td>QUAD</td>
<td>HEM</td>
</tr>
<tr>
<td>Bandwidth [GB/s] (CUPS NT / STREAM CUPS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRAM</td>
<td>342 / 418</td>
<td>333 / 388</td>
<td></td>
</tr>
<tr>
<td>MCDRAM</td>
<td>331 / 415</td>
<td>315 / 372</td>
<td></td>
</tr>
<tr>
<td>Bandwidth [GB/s] (Read)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRAM</td>
<td>77</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>MCDRAM</td>
<td>147 / 163</td>
<td>171</td>
<td></td>
</tr>
<tr>
<td>Bandwidth [GB/s] (Write)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRAM</td>
<td>33</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>MCDRAM</td>
<td>147 / 163</td>
<td>171</td>
<td></td>
</tr>
</tbody>
</table>

Cache Mode: Latency [ns] (Benchmark)

<table>
<thead>
<tr>
<th>Mode</th>
<th>Latency [ns] (Benchmark)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth [GB/s] (CUPS NT / STREAM CUPS)</td>
<td></td>
</tr>
<tr>
<td>DRAM</td>
<td>77</td>
</tr>
<tr>
<td>MCDRAM</td>
<td>147 / 163</td>
</tr>
<tr>
<td>Bandwidth [GB/s] (Read)</td>
<td></td>
</tr>
<tr>
<td>DRAM</td>
<td>77</td>
</tr>
<tr>
<td>MCDRAM</td>
<td>147 / 163</td>
</tr>
<tr>
<td>Bandwidth [GB/s] (Write)</td>
<td></td>
</tr>
<tr>
<td>DRAM</td>
<td>33</td>
</tr>
<tr>
<td>MCDRAM</td>
<td>147 / 163</td>
</tr>
</tbody>
</table>

The COSMO Code – 300k SLOC Fortran

Two main algorithmic motifs in dynamical core

Finite Difference Stencils

Tridiagonal Solvers

A principled approach to designing cache-to-cache broadcast algorithms

Multi-ary tree example

<table>
<thead>
<tr>
<th>Tree depth</th>
<th>Level size</th>
</tr>
</thead>
<tbody>
<tr>
<td>depth d = 2</td>
<td></td>
</tr>
<tr>
<td>k_j = 2</td>
<td></td>
</tr>
<tr>
<td>k_j = 3</td>
<td></td>
</tr>
</tbody>
</table>

Tree cost

Horizontal Diffusion Stencil Program tuned to Xeon Phi KNL

Work performed at the Intel Parallel Computing Center at ETH Zurich

Questions/Discussions?
Backup
Sorting in complex memories

Check! Let’s do something “Big Data”!

We’ll need a parallel sort on all cores, right?

Triad Cache Mode SNC4

Triad Flat Mode SNC4
Memory model: Bitonic Mergesort

- Slices of 16 elements go through a bitonic network.
- Communication: CPU₀ accesses data from local and remote caches.
- Synchronization: CPU₀ waits for CPU₁.
- Memory accesses: latency vs. bandwidth.
Modeling Bitonic Sorting

\[
C_{L1}(n) = \left\lceil \log_2(n) - 1 \right\rceil 2n \cdot \text{cost}_{L1} + 2n \cdot \text{cost}_{\text{mem}}
\]

\[
C_{L2}(n) = \frac{n}{n_{L1}} C_{L1}(n_{L1}) + \left( \left\lceil \log_2(n) - \log_2(n_{L1}) \right\rceil 2n \cdot \text{cost}_{L2} + \right.
\]

\[
C_{\text{mem}}(n) = \frac{n}{n_{L2}} C_{L2}(n_{L2}) + \left( \left\lceil \log_2(n) - \log_2(n_{L2}) \right\rceil 2n \cdot \text{cost}_{\text{mem}} \right)
\]
Bitonic Sort of 1 kiB

Synchronization outweights memory costs for small data!
So don’t parallelize too much!

(a) Sorting 1 KB of integers.
Bitonic Sort of 4 MiB

Latency (seconds)

Number of threads

- Mem. model Lat.
- Mem. model BW
- Full model Lat.
- Full model BW
- Measured

“parallelization boundary”

model including synchronization cost

model (just) memory costs
Bitonic Sort of 1 GiB

Always use all cores!

synchronization negligible
The most surprising result last ...

The model (and practical measurements) indicate that it does not matter.

Thesis: the higher bandwidth of MCDRAM did not help due to the higher latency ($\log^2 n$ depth).

Hey, but which memory? DRAM or MCDRAM?

Disclaimer: this is NOT the best sorting algorithm for Xeon Phi KNL. It is the best we found with limited effort. We suspect that a combination of algorithms will perform best.